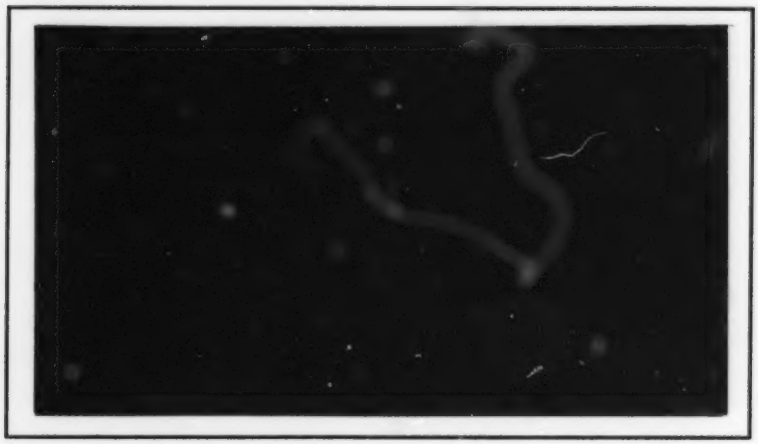


CENTRE FOR
SYSTEMS
SCIENCE

LCCR

Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 / (604) 291-4704

A Relevance Logic Characterization of Static Discontinuity Grammars

James Andrews, Veronica Dahl and Fred Popowich

CSS/LCCR TR91-12

December 1991

A Relevance Logic Characterization of Static Discontinuity Grammars[1]

James Andrews
Veronica Dahl
Fred Popowich

CMPT TR 91-10
LCCR TR 91-12
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada V5A 1S6

Abstract. A characterization of Static Discontinuity Grammars (SDGs), a logic grammar formalism due to Dahl, is given in this paper. A relevance logic sequent calculus proof system is given which is shown to be equivalent to SDGs for parsing problems, in the sense that a string of terminal symbols is accepted by a grammar if and only if the corresponding sequent is derivable in the calculus. One calculus is given for each of the two major interpretations of SDGs; the two calculi differ by only a small restriction in one rule.

The conceptual connection between relevance logic and grammar formalisms in general is also explored. Several linguistic formalisms, such as categorial grammars, share the "parsing as deduction" approach of logic grammars; relevance logic is shown to be a very useful semantic framework for these formalisms, since its bookkeeping about the order and occurrence of assumptions can be used to preserve information about the order and occurrence of words in text.

[1]It is the authors' wish that no agency should ever derive military benefit from the publication of this paper. Authors who cite this work in support of their own are requested to qualify similarly the availability of these results.

92-05826/

1 Introduction

The "logic grammar" framework has proven to be an effective tool in computational linguistics. Rather than offering specific linguistic theories themselves, logic grammars offer clear and logic-based ways to implement linguistic theories. Dahl's Static Discontinuity Grammars (SDGs) [Dah89, DP90] are a generalization of previous logic grammar formalisms such as Definite Clause Grammars (DCGs) [PW80], as well as of established grammar systems such as context-free grammars and Scattered Context Grammars [GH69].

Studies of the foundations of logic grammars have suggested that they have a relationship to relevance logic, the brand of logic which refuses to recognize an implication as valid unless the assumptions are relevant to the conclusion. This seems to be a fruitful area to explore, especially since relevance is also important to other areas of study in linguistics, such as the semantics of discourse [SW86].

This article describes exactly how SDGs are related to relevance logic, by giving a relevance-style logic which characterizes SDGs in a precise manner. First, however, let us give an informal explanation of how it comes to be that relevance logic is applicable to parsing as deduction.

1.1 Parsing as Deduction and its Difficulties

The parsing-as-deduction approach views the problem of parsing a sentence as the problem of deducing the assertion "we have heard (or read, or input) a sentence" from assertions of the form "we have heard (read, input) a certain word." For the problem of parsing a particular sentence, we encode the premisses and the conclusion in some formal logic, and then give axioms and/or rules of inference which allow all and only our desired sentences to be parsed. The particular linguistic theory being followed dictates the form of the axioms and the encoding.

Naturally, the encoding of the assumptions must include information about the occurrence and ordering of the words in the putative sentence. One possible way to encode, for instance, the parsing of the sentence "Evelyn loves books" is

$$\text{heard}([\text{evelyn}, \text{loves}, \text{books}]) \vdash \text{heard}(s)$$

Here, the assumptions have been encoded as the assertion that a list of three logical constants, "evelyn", "loves", and "books", have been heard; the conclusion has been encoded as the assertion that a sentence ("s") has been heard. The axioms of our logical system would then give us a way to parse the sentence by asserting that if we have heard certain forms of lists of tokens, we have heard corresponding lists of parts of speech, and so on.

In encoding a sentence as a list of tokens, though, we get the impression that we are not making the fullest possible use of the logical framework which we have adopted, since some of the grammar rules will involve non-linguistic concepts such as

the decomposition of lists. Logic grammars such as DCGs make use of this approach, though they generally hide details about lists from the programmer in standard sets of rules.

Say that we had instead encoded the example parsing as

$$\text{heard(evelyn)}, \text{heard(loves)}, \text{heard(books)} \vdash \text{heard(s)}$$

This style of encoding captures the fact that there are three separate events in the hearing, and will allow us to give simpler axioms that, for instance, assert that we have heard a proper name if we have heard "evelyn".

However, if we interpret $\vdash$ as the classical consequence relation, we have new problems, because this consequence relation is between a set of assumptions (possibly with some irrelevant elements) and a conclusion. For instance, if the above statement is provable, so should be

$$\text{heard(books)}, \text{heard(loves)}, \text{heard(evelyn)} \vdash \text{heard(s)}$$
$$\text{heard(evelyn)}, \text{heard(loves)}, \text{heard(loves)}, \text{heard(books)} \vdash \text{heard(s)}$$
$$\text{heard(evelyn)}, \text{heard(loves)}, \text{heard(very)}, \text{heard(books)} \vdash \text{heard(s)}$$

because the *set* of assumptions which allowed us to conclude that we had heard a sentence originally is still a subset of the set of assumptions in all three cases.

We could solve this problem by putting some temporal information explicitly into the logical encoding, by doing such things as labelling the words with the "times" at which they were heard. As with the solution of bundling words into a list, however, this solution may burden the axioms, rules of inference, and grammar rules with clumsy notational trivia.

To summarize, the parsing-as-deduction approach faces a basic problem in the logical encoding of parsing. To express needed information about the occurrence and ordering of words, some mechanism is needed to avoid the over-generation of the classical logic view.

1.2 Relevance Logic

Relevance logic [AB75, Rea88] seems to be useful in solving the problems associated with parsing as deduction, in part because it can express the needed occurrence and ordering information in a way that is less notationally burdened than the ways given above.

In proof-theoretic terms, the word-by-word encoding of a sentence into a list of assumptions has been foiled specifically by some unrestricted principles of deduction which are accepted under the classical consequence relation: the principles of permutation (which says that the order of assumptions is irrelevant), contraction (which says that we can duplicate any assumption), and weakening (which says that if we add assumptions we can still prove the same things).

However, these troublesome principles are largely rejected under relevance logic, which arises from the view that one should be able to deduce a conclusion from assumptions only if the assumptions are relevant to the conclusion and made in a relevant order. We can avoid having to deal with these principles by interpreting the assumptions as making up a sequence, rather than a set, and avoiding the introduction of axioms or rules of inference which allow the offending principles to creep back in unrestricted.

More evidence that the parsing-as-deduction approach works well in a relevance logic framework is provided by the literature, both the literature directly concered with logic grammars and that concerned with categorial grammar, another parsing-as-deduction grammar style. The sequent calculus version of the Lambek calculus for categorial grammars [Lam88] has clear connections with relevance logic. Linear logic [Gir87], a style of logic closely related to relevance logic (and which has indeed been claimed to be a variant of relevance logic), has also been shown to have connections to logic-based parsing. De Paiva [DP91] has proven the Lambek calculus to be a fragment of intuitionistic linear logic, and Hodas and Miller [HM91] have used linear logic to extend Definite Clause Grammars via a linear logic programming language. Bollen [Bol91] has also developed a relevance logic programming language, though without making any specific reference to linguistics.

1.3 The Approach of This Paper

This paper (like the examples in the Hodas and Miller paper described above) has its roots in "logic grammars", the study of using logic programming techniques for computational linguistics. The logic grammar framework it studies is Dahl's Static Discontinuity Grammars (SDGs), which grew out of Definite Clause Grammars [PW80] and Extraposition Grammars [Per81] in a desire to handle discontinuous constituents and "movement" at a more fundamental level. However, little will be said about computation here, other than to note that grammars can be encoded naturally in SDGs, and that SDGs can be computed with an acceptable degree of efficiency [DM90, DP90].

This paper shows that an important interpretation of SDGs can be characterized in a sound and complete manner by a relevance-logic *sequent calculus*, and that the other main interpretation of SDGs can be characterized by a simple restriction of that calculus. Since DCGs, Scattered Context Grammars, and so on are encompassed by the SDG formalism, this characterization also applies to them.

For instance, given a static discontinuity grammar G, the example sentence above would be parsed successfully if and only if the following sequent is a theorem of the sequent calculus:

$$\mathcal{F}(G), \langle evelyn, loves, books \rangle \vdash s$$

where $\mathcal{F}(G)$ is the translation of the grammar into relevance logic formulae. Note that the assumption that a word has been heard is represented simply by the word

itself; and that "intensional bunches" $\langle \ldots \rangle$ are used to group words, to give ordering information while allowing elements of bunches to be treated as separate assertions.

Section 2 of this paper gives an introduction to the syntax and interpretation of SDGs. Section 3 gives an introduction to the relevant aspects of sequent calculus, especially as they pertain to relevance logic. Section 4 gives the particular sequent calculus which will be used to characterize the SDG formalism, and proves the soundness and completeness results.

2 Static Discontinuity Grammars

Discontinuous grammars (DGs) [Dah89] were developed to allow the expression of relationships between components of a sentence which are distant from one another. The sequences of words in-between the distant components and not directly related to them are called "discontinuities", hence the name. Static discontinuity grammars (SDGs) are a restriction of DGs in which the discontinuities are not explicitly mentioned and moved around during the course of parsing.

The SDG formalism consists of a definition of grammar, and two related but distinct ways of interpreting grammars – that is, two ways of telling when a parse tree is admissible under the grammar. These two interpretations are known as the *rewriting* and the *tree admissibility* interpretations. Here we give definitions of the formalism and these interpretations which are based on those in [DP90], though expressed in a slightly different way for the present purposes.

2.1 Definitions

Definition 2.1 A *term* is an expression of the form $\mathbf{p}(\mathbf{t}_1, \ldots, \mathbf{t}_n)$, for $n \geq 0$, where $\mathbf{p}$ is a predicate name and each of $\mathbf{t}_1 \ldots \mathbf{t}_n$ is a constant, variable, or term. (We assume some given language of predicate names, constants, and variables, and we conflate terms and predicate applications.) When n is 0, we generally write the term as simply $\mathbf{p}$.

An *atom* is either a term (which will be interpreted as a non-terminal symbol of the grammar), a term within square brackets $[\mathbf{t}]$ (which will be interpreted as a terminal symbol), or the "empty list" expression $[\,]$. We will, when convenient, interpret a terminal as the list consisting of the single term, and $[\,]$ as the empty list of terms. We will often use $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, ... as meta-variables standing for arbitrary atoms.

A *clause* is an expression of the form $(\mathbf{t} \rightarrow \mathbf{A}_1, \ldots, \mathbf{A}_m)$, for $m \geq 1$, where $\mathbf{t}$ is a term and each of the $\mathbf{A}_i$'s is an atom[2].

An *SDG rule* is an expression of the form $\langle \mathbf{C}_1, \ldots, \mathbf{C}_n \rangle$, for $n \geq 1$, where each of the $\mathbf{C}_i$'s is a clause.

[2]Note that we do not allow m to be 0. Here we use the expression $\mathbf{t} \rightarrow [\,]$ to denote a clause with an empty body; as we will see later, this fits in better with the way in which we construct parse trees.

□

Examples. A simple grammar for sentences might be the following. (Variables are indicated by starting the name of the atom with a capital letter.)

$\langle\ s \rightarrow np(Subj), vt(Subj, Obj), np(Obj)\ \rangle$

$\langle\ np(person) \rightarrow [evelyn]\ \rangle$

$\langle\ np(thing) \rightarrow [books]\ \rangle$

$\langle\ vt(person, Any) \rightarrow [loves]\ \rangle$

This grammar generates the two sentences "Evelyn loves books" and "Evelyn loves Evelyn". It is in fact also a DCG, and if we left out the arguments to the nonterminals, it would also be a context-free grammar (although it would generate two more sentences). Note that each rule has only one clause in this example.

Rules with more than one clause are generally used for problems in which constituents are separated by intervening text, such as anaphora. Such problems are difficult to illustrate in small examples (see [Dah89] for larger ones). However, we can see their effect in small formal examples such as the following.

$\langle\ s \rightarrow as, bs\ \rangle$

$\langle\ as \rightarrow [\,]$
$\quad bs \rightarrow [\,]\ \rangle$

$\langle\ as \rightarrow [a(X)], as$
$\quad bs \rightarrow [b(X)], bs\ \rangle$

This example will generate a list of a's with arguments, followed by a list of b's with the same sequence of arguments. The exact set of sentences accepted by this grammar, however, is larger under the tree admissibility interpretation of SDGs than under the rewriting interpretation, as we will see.

2.2 The Rewriting Interpretation

The rewriting interpretation views an acceptable sentence or parse tree as the result of a series of rewritings of the initial symbol. There are two equivalent ways of characterizing this interpretation; one is a straightforward description of the rewriting process, and the other is a more graphical representation in terms of a parse tree.

Definition 2.2 Let R be the SDG rule $\langle(\mathbf{t}_1 \rightarrow \mathbf{A}_{11}, \ldots, \mathbf{A}_{1n_1}), (\mathbf{t}_2 \rightarrow \mathbf{A}_{21}, \ldots, \mathbf{A}_{2n_2}), \ldots, (\mathbf{t}_m \rightarrow \mathbf{A}_{m1}, \ldots, \mathbf{A}_{mn_m})\rangle$. Let Γ be the sequence of atoms

$$\Gamma_0,\ \mathbf{t}_1\theta,\ \Gamma_1,\ \mathbf{t}_2\theta,\ \Gamma_2,\ \ldots,\ \Gamma_{m-1},\ \mathbf{t}_m\theta,\ \Gamma_m$$

where each Γ_i is a (possibly empty) sequence of atoms, and θ is a substitution of terms for variables. Let Γ' be the sequence of atoms

$$\Gamma_0,\ \mathbf{A}_{11}\theta,\ldots,\mathbf{A}_{1n_1}\theta,\ \Gamma_1,\ \mathbf{A}_{21}\theta,\ldots,\mathbf{A}_{2n_2}\theta,\ \Gamma_2,\ \ldots,$$

$$\Gamma_{m-1},\ \mathbf{A}_{m1}\theta,\ldots,\mathbf{A}_{mn_m}\theta,\ \Gamma_m$$

(i.e., let Γ' be Γ with $\mathbf{t}_1$ replaced by $\mathbf{A}_{11},\ldots,\mathbf{A}_{1n_1}$, etc.) Then we say that $\Gamma \overset{R}{\Rightarrow} \Gamma'$ (in words, "Γ rewrites directly into Γ' by R").

We say that $\Gamma \Rightarrow \Gamma'$ ("Γ rewrites directly into Γ'") if there is some R such that $\Gamma \overset{R}{\Rightarrow} \Gamma'$.

We say that $\Gamma \overset{*}{\Rightarrow} \Gamma'$ ("Γ rewrites into Γ'") if Γ' can be obtained from Γ by zero or more direct-rewriting steps.

Under the rewriting interpretation, we say that a sequence of terminals Γ can be parsed as a non-terminal **s** in a grammar G if **s** rewrites into some sequence Γ' of terminals and empty lists, within which the sequence of terminals is Γ.

□

For instance, the sequence $[evelyn], [loves], [books]$ can be parsed as an s in the first example grammar because of the following sequence of rewritings:

$s \Rightarrow$ $np(person), vt(person, thing), np(thing)$
$\Rightarrow$ $[evelyn], vt(person, thing), np(thing)$
$\Rightarrow$ $[evelyn], [loves], np(thing)$
$\Rightarrow$ $[evelyn], [loves], [books]$

In the second example grammar, the sequence $[a(1)], [a(2)], [b(1)], [b(2)]$ can be parsed as an s because of the following rewritings:

$s \Rightarrow$ as, bs
$\Rightarrow$ $[a(1)], as, [b(1)], bs$
$\Rightarrow$ $[a(1)], [a(2)], as, [b(1)], [b(2)], bs$
$\Rightarrow$ $[a(1)], [a(2)], [\,], [b(1)], [b(2)], [\,]$

We can give another characterization of the rewriting interpretation by giving conditions on parse trees. This provides us with a more graphical representation of the parsing process, and also serves as the basis for the other major interpretation of SDGs. This "consistent parse tree" characterization will also be the most useful later, for proving the equivalence to the relevance logic.

Definition 2.3 A *quasi-parse tree* is a finitely-branching tree in which each interior node is labelled with a non-terminal, and each leaf node is labelled with any kind of atom (i.e., a terminal, non-terminal, or empty list [ ]).

The *fringe* of a labelled tree is the sequence of labels of its leaf nodes.

A *(complete) parse tree* is a quasi-parse tree whose fringe consists only of terminals and empty lists [ ].

An *indexing* of a quasi-parse tree T is T with each node indexed with an integer.

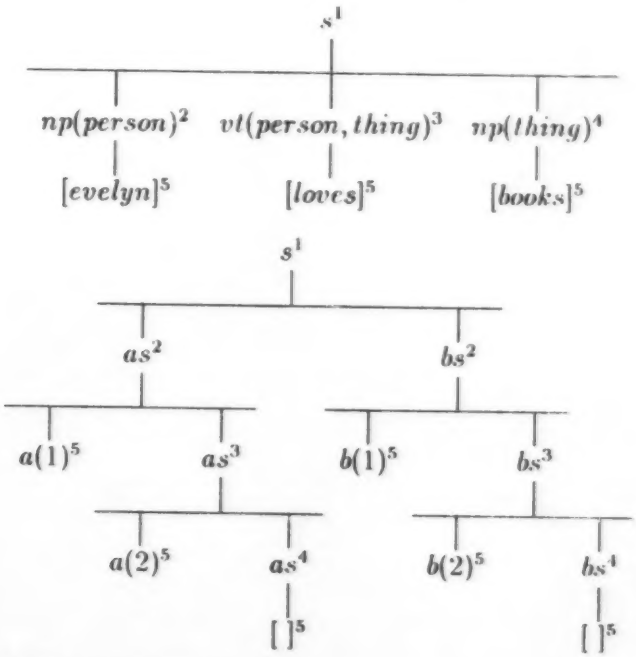
An indexed quasi-parse tree is *consistent (with rewriting)* if:

- Each node has a smaller index than that of any of its indexed child nodes; and
- For each i appearing as an index of an internal node in the tree, there is a rule in the grammar of the form $\langle(\mathbf{t}_1 \rightarrow \mathbf{A}_{11}, \ldots, \mathbf{A}_{1n_1}), (\mathbf{t}_2 \rightarrow \mathbf{A}_{21}, \ldots, \mathbf{A}_{2n_2}), \ldots, (\mathbf{t}_m \rightarrow \mathbf{A}_{m1}, \ldots, \mathbf{A}_{mn_m})\rangle$ and a substitution θ, such that the sequence of labels of nodes with index i is $\mathbf{t}_1\theta, \mathbf{t}_2\theta, \ldots, \mathbf{t}_m\theta$, and the sequence of labels of child nodes of each $\mathbf{t}_j$ node is $\mathbf{A}_{j1}\theta, \ldots, \mathbf{A}_{jn_j}\theta$.

By extension, an unindexed quasi-parse tree is consistent with rewriting if it has an indexing which is consistent with rewriting.

□

Note that the class of (complete) parse trees is a subset of the class of quasi-parse trees. The class of trees we are really interested in is the class of consistent complete parse trees. This definition leads to another characterization of when a sequence of terminals can be parsed by the rewriting interpretation, which is equivalent to the first definition; we must of course prove this, but first let us translate the examples above into indexed parse trees. (There is no unique indexing for a given consistent parse tree, since for instance any given integer can be added to all indices to create a new indexing; we give here only one possible indexing.)



Now, here is the equivalence theorem.

Theorem 2.4 A non-terminal **s** rewrites into a sequence of terminals and empty lists Γ iff there exists a consistent complete parse tree whose root node is **s** and whose fringe is Γ.

Proof. ($\rightarrow$) We must prove a more general theorem: if $\mathbf{s}$ rewrites into a sequence of atoms Δ, then there is a consistent indexed quasi-parse tree whose root node is $\mathbf{s}$ and whose fringe is Δ. We can do this by induction. In the base case, the indexed quasi-parse tree is the node $\mathbf{s}$ with index 1 (root node $\mathbf{s}$, fringe $\mathbf{s}$). In the inductive case, we have a consistent quasi-parse tree with root node $\mathbf{s}$ and fringe

$$\Gamma_0, \mathbf{t}_1\theta, \Gamma_1, \mathbf{t}_2\theta, \Gamma_2, \ldots, \Gamma_{m-1}, \mathbf{t}_m\theta, \Gamma_m$$

and we need to form a quasi-parse tree corresponding to

$$\Gamma_0, \mathbf{A}_{11}\theta, \ldots, \mathbf{A}_{1n_1}\theta, \Gamma_1, \mathbf{A}_{21}\theta, \ldots, \mathbf{A}_{2n_2}\theta, \Gamma_2, \ldots, \Gamma_{m-1}, \mathbf{A}_{m1}\theta, \ldots, \mathbf{A}_{mn_m}\theta, \Gamma_m$$

To do this, we let i be the maximum index appearing in the tree; we re-index each $\mathbf{t}_j$ node with $i+1$ (which must still be greater than the index of its parent); and to each $\mathbf{t}_j$ node we add child nodes $\mathbf{A}_{j1}\theta, \ldots, \mathbf{A}_{jn_j}\theta$ with index $i+2$. By the definition of rewriting-consistency, the quasi-parse tree is still consistent.

Note that the tree built up like this has all interior nodes labelled with even numbers and all leaf nodes with odd numbers. This is not important; it is only an artifact of how we have constructed the tree.

($\leftarrow$) Again, we must prove a more general theorem. First, let us assume that the indices in the indexed parse tree are $1 \ldots n$, and all leaf nodes have index n. (This does not lose generality because we can always re-index the parse tree to have this form.) Let us define the *k-th cut* in an indexed parse tree T as the sequence of nodes with index $\geq k$ which do not have a parent node with index $\geq k$. Therefore, if we imagine building up the tree in a stepwise fashion (at the k-th step adding all the children of the nodes with index k), then the k-th cut of the tree corresponds to the fringe of the tree before the k-th step. A node with index k whose parent has index j appears in all cuts from the $j+1$st to the k-th.

The more general theorem that we must prove is that $\mathbf{s}$ can rewrite to the k-th cut in the indexed parse tree. We can prove this by induction. Base case: the 1st cut is $\mathbf{s}$, and $\mathbf{s}$ does rewrite into itself. Inductive case: the $k+1$st cut differs from the k-th in that all the atoms corresponding to nodes with index k have been replaced by sequences of atoms with index $> k$. This replacement is exactly that effected by a direct rewriting step. Since the n-th cut in the parse tree is its fringe, $\mathbf{s}$ can rewrite into the fringe.

□

In conclusion, there are two different characterizations of the rewriting interpretations: one simple and straightforward one, and another which is less straightforward but more useful in some ways.

2.3 The Tree Admissibility Interpretation

There is another interpretation of SDGs, in which a grammar admits all the sentences it admitted under the rewriting interpretation, but may also admit some more. This

"tree admissibility" interpretation had two main motivations. Not necessarily in order, these were, firstly, that it was easier to implement an algorithm for this interpretation; and secondly, that it could be used to implement certain elements of linguistic theories such as GPSG [GKPS85].

The conditions under which a sequence of terminals is parsable as a given non-terminal under tree admissibility are expressed in terms of parse trees, as in the second characterization of the rewriting interpretation. Every parse tree which is consistent under rewriting is also admissible, and other parse trees are admissible which are not consistent under rewriting. However, there is no simple characterization of the tree admissibility interpretation in terms of rewriting.

Definition 2.5 An indexed quasi-parse tree is *admissible* if:

- No node dominates (is an ancestor of) any other node with the same index; and
- For each i appearing as an index of an internal node in the tree, there is a rule in the grammar of the form $\langle(\mathbf{t}_1 \rightarrow \mathbf{A}_{11}, \ldots, \mathbf{A}_{1n_1}), (\mathbf{t}_2 \rightarrow \mathbf{A}_{21}, \ldots, \mathbf{A}_{2n_2}), \ldots, (\mathbf{t}_m \rightarrow \mathbf{A}_{m1}, \ldots, \mathbf{A}_{mn_m})\rangle$ and a substitution θ, such that the sequence of labels of nodes with index i is $\mathbf{t}_1\theta, \mathbf{t}_2\theta, \ldots, \mathbf{t}_m\theta$, and the sequence of labels of child nodes of each $\mathbf{t}_j$ node is $\mathbf{A}_{j1}\theta, \ldots, \mathbf{A}_{jn_j}\theta$.

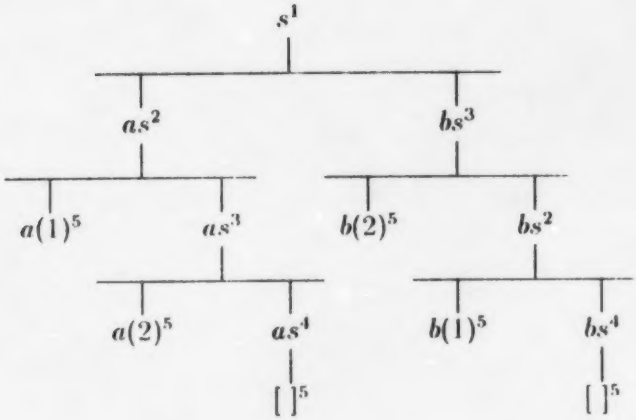
By extension, an unindexed quasi-parse tree is admissible if it has an indexing which is admissible.

Under the tree-admissibility interpretation, we say that a sequence of terminals Γ can be parsed as a non-terminal $\mathbf{s}$ in a grammar G if there is an admissible complete parse tree, within whose fringe the sequence of terminals is Γ.

□

This definition is exactly that for consistency with rewriting, except that the first condition has been relaxed: under admissibility, a node can have children which have lower indices – the only restriction is that they have no descendents with the same index.

For example, both of the example parse trees given above would be admissible parse trees, but so would the following:



Under the tree-admissibility interpretation, therefore, the sequence $[a(1)], [a(2)], [b(1)], [b(2)]$ can be parsed as an **s**, though it cannot under the rewriting interpretation.

In more straightforward but less precise terms, a tree is admissible if it is the result of taking an instance of a rule and patching its clauses over the tree in a left-to-right manner, taking another rule and doing the same thing, and so on. The difference between this interpretation and rewriting is that the clauses can be patched onto the tree *anywhere* relative to clauses from other rules.

2.4 Extensions of SDGs

For the application of SDGs to Government-Binding theory, Dahl [Dah89] defined two extensions to the basic theory as described here. Firstly, the bodies of clauses in SDG rules are allowed to contain calls to Prolog goals, which are called as conditions (or "constraints") on the application of the rule; and secondly, in every clause, the path of node names from the root of the parse tree to the node in the clause head is accessible in a path variable.

An SDG making use of either or both of these extensions can be translated back into an SDG as described here. We could have incorporated the extensions everywhere in this paper, but we felt that it might have turned out no simpler than leaving them out and giving the translation that follows.

We can translate any clause containing some Prolog constraint, written $\{\mathbf{A}\}$, into a simple clause by replacing it by the non-terminal **A**, and translating the Prolog rules for **A** into SDG rules. Each Prolog "fact", that is, clause of the form "**t.**", gets translated into the SDG rule $\langle \mathbf{t} \rightarrow [\,] \rangle$; while each Prolog "rule", that is, clause of the form "$\mathbf{t}$:- $\mathbf{A}_1, \ldots, \mathbf{A}_n$.", gets translated into the SDG rule $\langle \mathbf{t} \rightarrow \mathbf{A}_1, \ldots, \mathbf{A}_n \rangle$.

Similarly, we can translate an SDG making use of the path feature into a simple SDG by adding arguments to each clause. For every clause in every rule, we change the head of the clause from $\mathbf{p}(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ to $\mathbf{p}(\mathbf{t}_1, \ldots, \mathbf{t}_n, \mathbf{X})$, where **X** is some variable which does not appear anywhere in the grammar. We then change

every non-terminal $\mathbf{p'(t'_1,\ldots,t'_m)}$ in every clause body to a non-terminal of the form $\mathbf{p'(t'_1,\ldots,t'_m,[p(t_1,\ldots,t_n)|X])}$, where $\mathbf{p(t_1,\ldots,t_n,X)}$ is the head of the clause in which it appears. Each non-terminal will thus have had an argument added which passes down the path from root to current node. Every reference to the path variable could therefore be replaced by a reference to this new argument.

This is not to say that any *implementation* of SDGs could not handle these two extensions in some other, more efficient way. It is simply a translation which preserves the intended semantics while regaining the simple form of the original formulation.

3 Sequent Calculi

A *sequent calculus* is a particular kind of proof system with which one is able to formally deduce the truth of a kind of implicational expression called a "sequent". Since first defined by Gentzen [Gen69], sequent calculi have been used extensively to describe logical systems. They are particularly useful for describing relevance logics, for which implication is a central concept.

We shall be presenting our logical characterization of SDGs in the form of a sequent calculus. First, therefore, we should give some of the background concerning first order logic sequent calculi, and their relevance logic analogues.

3.1 First Order Sequent Calculi

A typical derivation in a first order sequent calculus is as follows.

$$\dfrac{\dfrac{\dfrac{}{q(3),p(3)\vdash p(3)} \quad \dfrac{}{q(3),p(3)\vdash q(3)}}{q(3),p(3)\vdash q(3)\vee r(3)}}{q(3),p(3)\vdash p(3)\&(q(3)\vee r(3))}$$
$$\dfrac{q(3)\&p(3)\vdash p(3)\&(q(3)\vee r(3))}{\forall x(q(x)\&p(x))\vdash p(3)\&(q(3)\vee r(3))}$$

As illustrated by this example, a sequent is an expression of the form $\mathbf{A_1,\ldots,A_n\vdash B}$, which is intended to be interpreted as "from the assumptions $\mathbf{A_1,\ldots,A_n}$ we can deduce $\mathbf{B}$." The calculus consists of a set of essentially syntactic rules which define what a valid derivation is. Some of the rules used in this derivation are:

&/l: $\dfrac{\Gamma,\mathbf{A_1},\mathbf{A_2}\vdash\mathbf{B}}{\Gamma,\mathbf{A_1}\&\mathbf{A_2}\vdash\mathbf{B}}$ &/r: $\dfrac{\Gamma\vdash\mathbf{B_1}\quad\Gamma\vdash\mathbf{B_2}}{\Gamma\vdash\mathbf{B_1}\&\mathbf{B_2}}$

Axiom: $\dfrac{}{\Gamma,\mathbf{A}\vdash\mathbf{A}}$ $\forall$/l: $\dfrac{\Gamma,\mathbf{A[x:=t]}\vdash\mathbf{B}}{\Gamma,\forall\mathbf{xA}\vdash\mathbf{B}}$

Note that the &/l and $\forall$/l rules have one upper sequent each, the &/r rule has two, and the axiom has none. The upper sequents are usually called *premisses* and the

lower sequent the *conclusion*; the group of formulae on the left-hand side are called the *antecedent*, and the formula on the right the *consequent*. All valid derivations must have axioms (applications of zero-premiss rules) at the top; the sequent that has been proven in a derivation is the conclusion of the bottommost rule application. In the example, we have concluded that $p(3)\&(q(3)\vee r(3))$ follows from the assumption that $\forall x(q(x)\&p(x))$.

Although the antecedent is written as a *sequence* of formulae, in first order logic we usually want to think of it as a *set*, having no associated sequencing or occurrence information. Thus when we write $\Gamma, \mathbf{A} \vdash \mathbf{A}$, we really mean any sequent whose conclusion appears in its antecedent. Another way of achieving the effect of sets is by maintaining a view of the antecedent as a sequence, and adding three rules which Gentzen called "structural rules":

Perm: $$\frac{\Gamma, \mathbf{A}_1, \mathbf{A}_2, \Gamma' \vdash \mathbf{B}}{\Gamma, \mathbf{A}_2, \mathbf{A}_1, \Gamma' \vdash \mathbf{B}}$$

Cont: $$\frac{\Gamma, \mathbf{A}, \mathbf{A}, \Gamma' \vdash \mathbf{B}}{\Gamma, \mathbf{A}, \Gamma' \vdash \mathbf{B}}$$

Thin: $$\frac{\Gamma, \mathbf{A}_1, \Gamma' \vdash \mathbf{B}}{\Gamma, \mathbf{A}_1, \mathbf{A}_2, \Gamma' \vdash \mathbf{B}}$$

The first rule, *permutation*, says that the order of assumptions is not important. The second, *contraction*, says that if we can conclude something from two copies of an assumption, we can conclude it from one. The third, *thinning*, says that if we conclude something from a set of assumptions, we can add new assumptions and still have a valid consequent. (The name comes from the fact that we can use the rule to "thin out" unimportant assumptions when trying to prove a given sequent.)

3.2 Relevance Logic

In relevance logic, these structural rules are not allowed with the full latitude that they are given in first order logic. Here we sketch how we can move calculi like the one given in the last section toward a relevance logic.

There are, of course, descriptions of the development of relevance logic from first philosophical principles [Rea88], to which readers are encouraged to refer for more of the philosophical motivation. The basic complaint that relevance logic raises with classical logic is that the notion of implication, $\mathbf{A} \rightarrow \mathbf{B}$, can result in nonsensical "truths". Since $\mathbf{A} \rightarrow \mathbf{B}$ is classically true iff it is not the case that $\mathbf{A}$ is true and $\mathbf{B}$ is false, sentences such as "if the moon is made of green cheese then Elvis is alive" and "if the continuum hypothesis is true then $2 + 2 = 4$" are classically true. The relevance logic alternative is to keep track of assumptions via the proof system so that no implication $\mathbf{A} \rightarrow \mathbf{B}$ is true merely because $\mathbf{A}$ is false, or because $\mathbf{B}$ is true regardless of $\mathbf{A}$. This leads to a treatment of implication that cannot be captured by the usual truth-table semantics given to connectives.

We can move toward a relevance logic by eliminating the structural rules, and insisting that only relevant assumptions be used in axioms:

Axiom: $$\frac{}{\mathbf{A} \vdash \mathbf{A}}$$

But we soon run into problems. For instance, we cannot prove as simple a sequent as $\mathbf{A}, \mathbf{B} \vdash \mathbf{A}\&\mathbf{B}$, because both premisses will have an irrelevant assumption. Yet it is clear that we should be able to deduce the consequent from the antecedent of this sequent, even if we insist that assumptions be relevant.

To regain the derivability of this sequent, what we need is a conjunction which splits the list of assumptions in two, giving one part to the left-hand premiss and the other to the right-hand. This conjunction is written $\otimes$ in the literature, and is called "intensional conjunction". The rule for when it appears in the antecedent is the same, but its consequent rule is as follows:

$\otimes$/r: $$\frac{\Gamma \vdash \mathbf{B}_1 \quad \Gamma' \vdash \mathbf{B}_2}{\Gamma, \Gamma' \vdash \mathbf{B}_1 \& \mathbf{B}_2}$$

Similar problems arise associated with the loss of permutation and thinning. With permutation, we find that we want not an absolute banishment, but a restriction of the principle. We could re-introduce permutation by introducing a new, unary connective κ [Yet90], which allows the formula so annotated to move around in the antecedent; the solution we will adopt here, however, is to allow formulae in the antecedent to be grouped into "bunches" [Rea88], where bunches can be permuted but the formulae within a given bunch cannot. As we will see, the bunches we will use will actually behave in a way which is unique (in our knowledge) with respect to the $\otimes$ conjunction, but which still preserves the intent of the bunching operation.

Bunches are a feature of traditional relevance logic, but to regain some form of thinning, we will also use a feature of linear logic which has a questionable relationship to traditional relevance logic: the unary "of course" connective, "!" [Gir87]. In linear logic, "!" formulae (and only those formulae) can be duplicated at will:

Cont/l: $$\frac{\Gamma, !\mathbf{A}, !\mathbf{A}, \Gamma' \vdash \mathbf{B}}{\Gamma, !\mathbf{A}, \Gamma' \vdash \mathbf{B}}$$ Der/l: $$\frac{\Gamma, \mathbf{A}, \Gamma' \vdash \mathbf{B}}{\Gamma, !\mathbf{A}, \Gamma' \vdash \mathbf{B}}$$

"!" formulae are therefore used as a source of copies of selected assumptions. To capture the fact that a "!" formula can stand for zero copies, these formulae are often allowed in axioms:

Axiom: $$\frac{}{!\Gamma, \mathbf{A} \vdash \mathbf{A}}$$

We can look at "!" assumptions as "background" or "generic" assumptions, which we accept as general knowledge and whose relevance to conclusions we are not concerned

about.

Intuitively, the importance of these relevance-logic concepts and connectives to Static Discontinuity Grammars in particular is as follows. The intensional conjunction $\otimes$, together with the notion of bunching, allows us to express the sequencing information we need (like the order of terms within clauses and the order of clauses within rules), the allocation of the various clauses of a rule over the sentence being parsed, and the movement of groups of rules past discontinuities. The "!" connective allows us to then encode the grammar rules as generic assumptions, exempt from the rigorous bookkeeping we must do on the other assumptions.

In fact, since sequencing information is an important component of many linguistic theories, we would expect that relevance logic could be generally useful in describing their underlying structure. For instance, in Immediate Dominance/Linear Precedence (ID/LP) grammars, a rewriting ("immediate dominance") rule specifies only the symbols in the sequence replacing a symbol, but not their order; separate "linear precedence" rules specify constraints on the possible orders. Relevance logics could conceivably be used to illuminate the logical structure of such grammars as well.

In summary, concepts important to the relevance logic point of view can be presented simply by the use of sequent calculi. A relevance logic sequent calculus can be obtained by weakening a standard first-order sequent calculus, and then re-introducing, in a restricted way, some of the discarded principles. For the purpose of characterizing SDGs, we will be re-introducing these principles in a particular way which will make clear the relationship between them and relevance logic.

4 The Relevance Logic Characterization

We wish to characterize SDGs with logic, specifically a relevance logic. What this means, essentially, is that we must give a logical description of the set of sentences which is accepted by a grammar, under the rewriting interpretation and under the tree admissibility interpretation.

In this section, we do this by means of a relevance logic sequent calculus. Given a grammar and a particular sentence to parse, we form a sequent to be proven. Using the rules of a sequent calculus (two calculi are defined here, one for rewriting and one for tree admissibility), we then try to derive the sequent. As we will prove, the sequent is derivable if and only if the sentence is acceptable; the method for forming the sequent, together with the rules of the proof system, can therefore be taken as a characterization of SDGs.

Bollen, in his description of a relevance logic programming language [Bo91], also gives a sequent calculus as the characterization of its logical structure. This is partially because the proof theory of relevance logic is clearer than the model theory, but Bollen's work also fits into a minor tradition of using sequent calculus to characterize logic programming systems [HS84, HSH90]. As SDGs arose from the logic grammar

framework, which themselves arose from general logic programming, Bollen's work has clear connection with ours.

One principle we will try to follow here is that of making the sequent calculus encode fairly general principles of reasoning. We do this in order to arrive at a system which can truly be said to be a logical view of the problem, and not simply a reworking of the original definitions.

The following subsections present the method of translation of a parsing problem into a sequent; the rules of the sequent calculi, and a brief discussion thereof; the proof of completeness (that every acceptable sentence corresponds to a derivation); the proof of soundness (that every derivation corresponds to an acceptable sentence); and finally a discussion of the relationship between the sequent calculi we have defined and the relevance logics in the literature.

4.1 Translation into Formulae

Our intent is to give a sequent calculus in which SDG rules can act as assumptions, on the left-hand side of a sequent to be proven. To do this, we must give the translation of SDG rules into formulae.

Definition 4.1 We define the *formulaic versions* of clauses, rules, and grammars as follows. $\mathcal{F}(e)$ denotes the formulaic version of an expression e.

- Let C be a clause, $\mathbf{t} \rightarrow \mathbf{A}_1, \ldots, \mathbf{A}_m$, where $m \geq 1$. $\mathcal{F}(C)$ is the relevance logic formula $\mathbf{t} \leftarrow (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_m)$.
- Let R be an SDG rule, $\langle C_1, \ldots, C_n \rangle$, where $n \geq 1$. $\mathcal{F}(R)$ is the relevance logic formula $!(\forall[\mathcal{F}(C_1) \otimes \cdots \otimes \mathcal{F}(C_n)])$, where $\forall[\mathbf{B}]$ is the universal closure of $\mathbf{B}$.
- Let G be an SDG grammar, consisting of rules $R_1 \ldots R_n$, where $n \geq 1$. $\mathcal{F}(G)$ is the sequence of formulae $\mathcal{F}(R_1) \ldots \mathcal{F}(R_n)$.

□

Readers familiar with logic programming will note that SDG rules are thus being interpreted in much the same way as Horn clauses are in the standard semantics of logic programs [Llo84]. The $\otimes$ connective is just the "intensional" form of conjunction which relevance logic has in contrast to the extensional "&" of classical logic. SDG rules, because we wish them to be re-usable, movable background assumptions, have the prefix "!", denoting genericity.

As we will see later, however, the sentence to be parsed, as well as individual instances of rules, will appear as contingent assumptions, grouped into bunches of formulae which must stay in the order in which they have been placed and cannot be duplicated.

Definition 4.2 An *(intensional) bunch* is an expression of the form $\langle \mathbf{B}_1, \ldots, \mathbf{B}_n \rangle$, where each $\mathbf{B}_i$ is a formula.

A *sequent* (for the present purposes) is an expression of the form

$$!\Gamma, \Delta \vdash \mathbf{B}$$

where $!\Gamma$ is a sequence of formulae preceded by "!", Δ is a sequence of intensional bunches, and $\mathbf{B}$ is a formula.

□

We will generally use boldface letters to denote formulae, using $\mathbf{R}$ for the formulaic version of a rule, $\mathbf{C}$ for that of a clause, $\mathbf{A}$ for an atom, $\mathbf{t}$ for a term, and $\mathbf{B}$ for any formula. (Note that we use R and C for the original versions of rules and clauses.) We will use $!\Gamma$ to denote a sequence of formulae all of which are preceded by !, Δ to denote a sequence of intensional bunches, and Λ to denote a sequence of formulae; each may denote the empty sequence. All of these meta-variables (as they are called) may be subscripted ($\mathbf{A}_1$, $\mathbf{A}_2$, etc.) or primed ($\mathbf{A}'$).

A particular parsing problem, then, is translated into a sequent as follows. Given a grammar G and a sequence of terminals $\mathbf{t}_1, \ldots, \mathbf{t}_k$ to be parsed as a non-terminal $\mathbf{s}$, the corresponding sequent is

$$\mathcal{F}(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}$$

This sequent says that, taking the grammar rules as generic assumptions, and the "heard" words as a sequence of relevant assumptions in which order is important, we can infer that we have heard an $\mathbf{s}$. We want it to be the case that the sequent will be derivable in our sequent calculus if and only if the sequence of words can be parsed as $\mathbf{s}$ under the tree admissibility interpretation. We must, therefore, give rules which will allow this to be the case, and must prove that it is the case.

4.2 Sequent Calculus Rules

The sequent calculus rules we shall be considering are contained in Figure 1. These rules define the calculus Rsa ("Relevance logic for SDGs under Admissibility"), and can also be restricted slightly to define Rsr ("Relevance logic for SDGs under Rewriting").

Each of the rules can be interpreted in two ways. The first way is to look at it as a rule for introducing (into the lower, conclusion sequent) a particular connective, modality, or form of expression on the right-hand or left-hand side of the sequent. This is the traditional way of looking at such rules in logic, and the names appearing at the side of the rule definitions reflect this view.

The other way to look at a rule is as a method of solving a particular parsing problem or sub-problem; given the lower, conclusion sequent, the rule gives a way of constructing a derivation of it. This view is reflected in the subtitles under each rule.

Cont/l: $$\frac{!\Gamma, !\mathbf{R}, !\Gamma', \Delta, \langle \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}{!\Gamma, !\mathbf{R}, !\Gamma', \Delta, \Delta' \vdash \mathbf{B}}$$

("copy SDG rule")

$\forall$/l: $$\frac{!\Gamma, \Delta, \langle \mathbf{R}[\mathbf{x} := \mathbf{t}] \rangle, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle \forall \mathbf{x} \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}$$

("choose substitutions")

$\otimes$/l: $$\frac{!\Gamma, \Delta, \langle \Lambda, \mathbf{C}, \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle \Lambda, \mathbf{C} \otimes \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}$$

("separate rule clauses")

$\otimes$/r: $$\frac{!\Gamma, \langle \Lambda_1 \rangle, \ldots, \langle \Lambda_k \rangle \vdash \mathbf{B} \quad !\Gamma, \langle \Lambda'_1 \rangle, \ldots, \langle \Lambda'_k \rangle \vdash \mathbf{B}'}{!\Gamma, \langle \Lambda_1, \Lambda'_1 \rangle, \ldots, \langle \Lambda_k, \Lambda'_k \rangle \vdash \mathbf{B} \otimes \mathbf{B}'}$$

("allocate clauses across tree")

$\leftarrow$/l: $$\frac{!\Gamma, \Delta, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle \mathbf{t} \leftarrow \mathbf{B} \rangle, \Delta' \vdash \mathbf{t}}$$

("call clause")

Refl: $$\frac{}{!\Gamma, \langle \mathbf{t} \rangle \vdash [\mathbf{t}]}$$

("match terminal")

$\langle\rangle$/l: $$\frac{!\Gamma, \Delta, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle\rangle, \Delta' \vdash \mathbf{B}}$$

("ignore empty rule")

[ ]/r: $$\frac{}{!\Gamma \vdash [\,]}$$

("match empty sequence")

Figure 1: Rules for sequent calculus Rsa.

The calculus Rsr consists of exactly the same rules as Rsa, with the exception that in the $\leftarrow$/l rule, the first sequence Δ of intensional bunches must be *empty*. That is, in Rsr the $\leftarrow$/l rule is replaced by the rule:

$$\leftarrow/\mathrm{l}: \qquad \frac{!\Gamma, \Delta' \vdash \mathbf{B}}{!\Gamma, \langle \mathbf{t} \leftarrow \mathbf{B} \rangle, \Delta' \vdash \mathbf{t}}$$

Thus, the sequence of bunches acts like a list into which a bunch can be placed at any point, but from which only the topmost element can be popped.

There are other ways in which we could have presented Rsa and Rsr. In particular, consider the calculus in which all of the rules Cont/l, $\forall/l$, $\otimes/l$, $\leftarrow$/l, and $\langle\rangle$/l have the above restriction of Δ to the empty sequent. It is provable that the resulting sequent calculus would be equivalent to Rsr, but Rsa could be regained by adding a rule of permutation of bunches:

$$\text{Perm/l}: \qquad \frac{!\Gamma, \Delta, \Delta_1, \Delta_2, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \Delta_2, \Delta_1, \Delta' \vdash \mathbf{B}}$$

This would be the equivalent of treating the antecedent as a multiset rather than a sequence.

The present formulation was chosen because it seemed to provide the simplest framework for proving the soundness and completeness theorems.

4.3 Completeness

This section contains the proof of completeness of the characterization: that is, the proof that every admissible parse tree (respectively, rewriting-consistent parse tree) corresponds to a derivation in Rsa (resp. Rsr). This is the first part of the demonstration that these proof systems are an effective characterization of SDGs; the converse, soundness, is shown in the next section.

The completeness proof basically consists of building up a partial derivation from the given tree, proving that the resulting bunches on the left of the bottommost sequent in the derivation are all instances of rules, and, finally, extending the derivation to the desired conclusion.

The formal statement of the theorem is as follows:

> Given a grammar G, if a sequence of terminals $[\mathbf{t}_1] \cdots [\mathbf{t}_k]$ can be parsed as a non-terminal $\mathbf{s}$ under the tree admissibility (resp. rewriting) interpretation, then there is a derivation in Rsa (resp. Rsr) for the sequent
>
> $$\mathcal{F}(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}$$

Before we can prove this, however, we must make some definitions and prove some auxiliary lemmata. We will not need to give two separate proofs for the two interpre-

tations; instead, we will cast the proof in terms of the tree-admissibility interpretation, occasionally making remarks specific to the rewriting interpretation.

We can associate an *analogue derivation* to any subtree of an admissible (indexed) parse tree T; in general, we will have to define the analogue derivation of any sequence of subtrees of T, although the only derivations of this type we will be interested in are those of sequences of sibling subtrees. Each bunch in the antecedents of the sequents of the analogue derivation will be indexed by an integer, just like the nodes in T. These indices will be for accounting purposes only, and will not affect the validity of the derivation. A bunch $\langle \Lambda \rangle$ indexed by k will be written $\langle \Lambda \rangle^k$. (In the rest of this subsection, as we did in the proof of Theorem 2.4, we will assume without loss of generality that the leaf nodes of the indexed parse tree are all indexed with the highest index in the tree.)

Definition 4.3 The *analogue derivation* of a sequence $S_1, \ldots, S_k$ of subtrees of an admissible indexed parse tree T is defined as follows.

- If the sequence is the single subtree consisting of the single node [ ] indexed by m, then the analogue derivation is simply

$$\frac{\overline{\mathcal{F}(G) \vdash [\,]}}{\mathcal{F}(G), \langle\rangle^m \vdash [\,]}$$

- If the sequence is the single subtree consisting of the single node [t] indexed by m, then the analogue derivation is

$$\overline{\mathcal{F}(G), \langle \mathbf{t} \rangle^m \vdash [\mathbf{t}]}$$

- If the sequence is $S_1, \ldots, S_k$ where $k > 1$, then: let $\mathcal{D}_1$ be the analogue derivation of S_1, and $\mathcal{D}_2$ be the analogue derivation of $S_2, \ldots, S_k$. If there are indices of bunches in $\mathcal{D}_2$ which do not appear in $\mathcal{D}_1$, then extend $\mathcal{D}_1$ by performing applications of $\langle\rangle$/l which insert empty bunches with those indices into the antecedent, so that the bunches are in ascending order of index. Call the resulting derivation $\mathcal{D}_1'$; extend $\mathcal{D}_2$ to $\mathcal{D}_2'$ in a similar manner. Then the analogue derivation of $S_1, \ldots, S_k$ is formed from $\mathcal{D}_1'$ and $\mathcal{D}_2'$ by using the $\otimes$/r rule:

$$\frac{\begin{array}{cc} (\mathcal{D}_1) & (\mathcal{D}_2) \\ \vdots & \vdots \\ !\Gamma, \langle \Lambda_1 \rangle^1, \ldots, \langle \Lambda_k \rangle^k \vdash \mathbf{B} & !\Gamma, \langle \Lambda_1' \rangle^1, \ldots, \langle \Lambda_k' \rangle^k \vdash \mathbf{B}' \end{array}}{!\Gamma, \langle \Lambda_1, \Lambda_1' \rangle^1, \ldots, \langle \Lambda_k, \Lambda_k' \rangle^k \vdash \mathbf{B} \otimes \mathbf{B}'}$$

- Finally, if S is the single subtree whose root node is the non-terminal t with index i, then: let $\mathcal{D}$ be the analogue derivation of the sequence $S_1, \ldots, S_k$ of children of the root node of S. The analogue derivation of S is $\mathcal{D}$ extended by an application of $\leftarrow$/l, with the new bunch put in its proper place in the order of bunches:

$$\frac{\begin{array}{c}\mathcal{D}\\ \vdots\\ !\Gamma, \langle\Lambda_1\rangle^1, \ldots, \langle\Lambda_{i-n_1}\rangle^{i-n_1}, \langle\Lambda_{i+n_2}\rangle^{i+n_2}, \ldots, \langle\Lambda_m\rangle^m \vdash \mathbf{B}\end{array}}{!\Gamma, \langle\Lambda_1\rangle^1, \ldots, \langle\Lambda_{i-n_1}\rangle^{i-n_1}, \langle \mathbf{t} \leftarrow \mathbf{B}\rangle^i, \langle\Lambda_{i+n_2}\rangle^{i+n_2}, \ldots, \langle\Lambda_m\rangle^m \vdash \mathbf{t}}$$

□

The analogue derivation of a particular parse tree is certainly a valid derivation; however, we have not yet shown that it will lead to the sequent corresponding to the parsing problem. We need to show that each of the bunches in the antecedent of the conclusion actually corresponds to a rule, and is not just a random collection of formulae. To do this, it suffices to prove the following properties of the analogue derivation.

Lemma 4.4
(1) Let $S_1, \ldots, S_k$ be a sequence of subtrees of an indexed admissible parse tree. Then the analogue derivation to that sequence will have the consequent $\mathbf{A}_1 \otimes \ldots \otimes \mathbf{A}_k$, where each $\mathbf{A}_i$ is the root node of S_i.
(2) Let $S_1, \ldots, S_k$ be a sequence of subtrees of an indexed admissible parse tree. Then every index of a bunch in its analogue derivation will appear in one of the S_i's.
(3) Let $S_1, \ldots, S_k$ be a sequence of subtrees of an indexed admissible parse tree. Let $\mathbf{t}_1, \ldots, \mathbf{t}_j$ be the sequence of nodes in $S_1, \ldots, S_k$ which have index i. Let $\mathbf{A}_{n1}, \ldots, \mathbf{A}_{nm_n}$ be the sequence of children of $\mathbf{t}_n$. Then there will be only one bunch with index i in the conclusion of the analogue derivation, and it will be $\langle \mathbf{t}_1 \leftarrow \mathbf{A}_{11} \otimes \ldots \otimes \mathbf{A}_{1m_1}, \ldots, \mathbf{t}_j \leftarrow \mathbf{A}_{j1} \otimes \ldots \otimes \mathbf{A}_{jm_j} \rangle$.
(4) Except for the bunch with highest index, each bunch $\langle \mathbf{C}_1, \ldots, \mathbf{C}_k \rangle$ in the conclusion of the analogue derivation of an admissible parse tree is of the form $\langle \mathcal{F}(C_1), \ldots, \mathcal{F}(C_k) \rangle$, where $\langle C_1, \ldots, C_k \rangle$ is a rule of the grammar.
(5) If T is a rewriting-consistent parse tree, then the analogue derivation to T is a valid derivation in Rsr.
(6) The bunch with the highest index in the conclusion of the analogue derivation of an admissible parse tree is of the form $\langle \mathbf{t}_1, \ldots, \mathbf{t}_n \rangle$, where $\mathbf{t}_1, \ldots, \mathbf{t}_n$ is the sequence of terminals in the fringe of the parse tree.

Proof. (1) is easily proven by induction on the structure of the parse tree, as is (2).

(3) is proven by induction as well. The only difficult case is for a single subtree with a non-terminal at its root. The new bunch has a unique index because by the tree admissibility condition, no node other than the root node of the subtree will have the index of that node, and by (2) that index will not appear in the derivation. The new bunch is of the proper form because by (1) the body of the new clause will consist of the sequence of children of the root.

(4) follows from the fact that the sequence of nodes with the same index (together with their children) must correspond to a rule in the grammar.

(5) follows from the fact that by the definition of a rewriting-consistent parse tree, every node will have a lower index than any of its descendents has; and therefore the bunch introduced by the application of $\leftarrow$/l will be at the left of the sequence of indexed bunches.

(6) is easily proven by induction.

□

We are now in a position to prove the completeness theorem.

Theorem 4.5 (Completeness) Given a grammar G, if a sequence of terminals $[\mathbf{t}_1]\cdots[\mathbf{t}_k]$ can be parsed as a non-terminal $\mathbf{s}$ under the tree admissibility (resp. rewriting) interpretation, then there is a derivation in Rsa (resp. Rsr) for the sequent

$$\mathcal{F}(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}$$

Proof. From the assumption, we have an admissibie (resp. rewriting-consistent) indexed parse tree T, whose fringe consists of the terminals $[\mathbf{t}_1]\cdots[\mathbf{t}_k]$ in that order, possibly with some empty sequences [ ] in-between. From the lemma, we therefore have a derivation in Rsa (resp. Rsr), all of whose bunches correspond to rules except for the last one, which is the bunch $\langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle$.

We can therefore extend the analogue derivation by a sequence of $\otimes$/l applications (to form each rule bunch into a single formula), followed by a sequence of $\forall$/l applications (to reconstruct the formulaic version of each rule), followed by a sequence of Cont/l applications (to contract each copy of each rule). The end of the derivation will look like this:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathcal{F}(G), \langle \mathbf{C}_{11}, \ldots, \mathbf{C}_{1m_1} \rangle, \ldots, \langle \mathbf{C}_{n1}, \ldots, \mathbf{C}_{nm_n} \rangle, \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}}}{\text{(some } \otimes\text{/l steps)}}}{\mathcal{F}(G), \langle \mathbf{R}_1\theta \rangle, \ldots, \langle \mathbf{R}_n\theta \rangle, \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}}}{\text{(some } \forall\text{/l steps)}}}{\mathcal{F}(G), \langle \forall[\mathbf{R}_1] \rangle, \ldots, \langle \forall[\mathbf{R}_n] \rangle, \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}}}{\text{(some Cont/l steps)}}}{\mathcal{F}(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}}$$

which is indeed the sequent we wanted to prove derivable.

□

4.4 Soundness

We have now proven that each parse tree corresponds to a derivation. However, we have not yet proven that each derivation corresponds to a parse tree. We need to do this in order to show that our logic is not stronger than it needs to be; otherwise, there would be nothing to distinguish this logic from, say, conventional first order logic as a characterization of SDGs.

The statement of the soundness theorem is essentially the converse of that of the completeness theorem:

> Given a grammar G, if there is a derivation in Rsa (resp. Rsr) for the sequent
>
> $$\mathcal{F}(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}$$
>
> then the sequence of terminals $[\mathbf{t}_1] \cdots [\mathbf{t}_k]$ can be parsed as the non-terminal $\mathbf{s}$ under the tree admissibility (resp. rewriting) interpretation.

The proof of soundness will also be, in some sense, the converse of the proof of completeness. Whereas in the completeness proof we took a parse tree, built a partial derivation, and then extended the derivation to the desired conclusion, in the soundness proof we will take a derivation, put it in a "normal form" which will allow us to ignore steps at the bottom, and build a parse tree from the top of the derivation.

We will begin by defining this very general notion of normal form, and proving that every derivation has a normal form. These definitions apply, and the properties hold, for all derivations, not just the subset of derivations corresponding to parsing problems. For instance, even if we had a sequent with a random mixture of atoms, implications, and conjunctions in the bunches of the antecedent, it would still have a normal form. We do this to emphasize, and take advantage of, the fact that general logical principles are being used in the proof system, rather than principles specific to the application we have in mind.

Definition 4.6 We say that a derivation (in either Rsa or Rsr) is in *normal form* if the sequence of rules applied, reading upward from the bottommost step, consists of 0 or more applications of Cont/l, followed by 0 or more applications of $\forall$/l and $\otimes$/l.

□

Generally, in proof theory, when we define the notion of a "normal form", we do so with a view to proving that every derivation has a corresponding normal form which is in some way equivalent. That is the case here.

Lemma 4.7 (Normalization) For every derivation $\mathcal{D}$ (in Rsa or Rsr), there is a derivation in normal form which has the same conclusion.

Proof. The proof will be in two stages. The first stage will result in a derivation with the same conclusion but with all its Cont/l steps at the very bottom. The second stage will similarly put the $\forall$/l and $\otimes$/l applications in their proper places.

First stage: we can measure how far a derivation is from the proper form by the rational number $j + \frac{k}{k+1}$, where j is the number of applications of Cont/l which are higher than some application of another rule, and k is the height (in the derivation) of the lowest such rule application[3]. Note that we can lower this measure either by

[3]Readers familiar with transfinite induction might prefer to view the measure as $(j \cdot \omega) + k$ instead; this will have the same effect.

keeping j constant and lowering k, or by lowering j regardless of k. We can do this, in turn, by taking the lowest application of Cont/l which is higher than some application of another rule, and moving it lower in the derivation.

The argument therefore goes as follows. Given any derivation $\mathcal{D}$ with a non-zero measure, we can build a derivation with the same conclusion but a lower measure. Consider the lowest application of Cont/l which is higher than some application of another rule. There are two subcases.

If the conclusion of this application is the premiss of a $\forall$/l, $\otimes$/l, $\leftarrow$/l, or $\langle\rangle$/l rule, then the relevant portion of the derivation will look like this:

$$\dfrac{\dfrac{\dfrac{\vdots}{!\Gamma, \Delta, \langle \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}}{!\Gamma, \Delta, \Delta' \vdash \mathbf{B}}\ \text{(by Cont/l)}}{!\Gamma, \Delta'', \Delta''' \vdash \mathbf{B}'}\ \text{(by other rule)}$$
$$\vdots$$

The new derivation with the lower measure will then look like this:

$$\dfrac{\dfrac{\dfrac{\vdots}{!\Gamma, \Delta, \langle \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}}{!\Gamma, \Delta'', \langle \mathbf{R} \rangle, \Delta''' \vdash \mathbf{B}'}\ \text{(by other rule)}}{!\Gamma, \Delta'', \Delta''' \vdash \mathbf{B}'}\ \text{(by Cont/l)}$$
$$\vdots$$

The only other subcase is when the conclusion of the application of Cont/l is one of the premisses of an application of $\otimes$/r. If it is the left-hand premiss, the derivation will look like this:

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{!\Gamma, \Delta_1, \langle \mathbf{R} \rangle, \Delta_2 \vdash \mathbf{B}}}{!\Gamma, \Delta_1, \Delta_2 \vdash \mathbf{B}}\ \text{(by Cont/l)} \qquad \dfrac{\mathcal{D}'}{!\Gamma, \Delta_1', \Delta_2' \vdash \mathbf{B}'}}{!\Gamma, \Delta_1'', \Delta_2'' \vdash \mathbf{B} \otimes \mathbf{B}'}\ \text{(by } \otimes\text{/l)}$$
$$\vdots$$

The new derivation with the lower measure will then look like this:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}}{!\Gamma, \Delta_1, \langle \mathbf{R} \rangle, \Delta_2 \vdash \mathbf{B}} \qquad \dfrac{\mathcal{D}'}{!\Gamma, \Delta_1', \Delta_2' \vdash \mathbf{B}'}}{!\Gamma, \Delta_1', \langle\rangle, \Delta_2' \vdash \mathbf{B}'}\ \text{(by } \langle\rangle\text{/l)}}{!\Gamma, \Delta_1'', \langle \mathbf{R} \rangle, \Delta_2'' \vdash \mathbf{B} \otimes \mathbf{B}'}\ \text{(by } \otimes\text{/l)}}{!\Gamma, \Delta_1'', \Delta_2'' \vdash \mathbf{B} \otimes \mathbf{B}'}\ \text{(by Cont/l)}$$
$$\vdots$$

Similar reasoning can be followed when the conclusion of the Cont/l application is the *right*-hand premiss of $\otimes$/l.

In all subcases, k will be lower, unless the transformation caused the rule application to sink to the bottom (or the sequence of Cont/l steps at the bottom) – in which case k will be higher, but j will be lower because the lowered step will no longer contribute to that number. The transformation will also preserve the conclusion of the whole derivation, because the transformation preserves the conclusion of the pertinent sequence of rules unchanged. We can repeat this process until the measure becomes equal to zero, at which point there will be no applications of Cont/l except those at the bottom.

Second stage: We can repeat the reasoning of the first stage to move all applications of $\forall$/l and $\otimes$/l to their destination just above the applications of Cont/l. This stage is actually easier because we do not have to make a special case for $\otimes$/r; all cases work as in the first subcase of stage 1.

□

Note that in derivations in general we cannot always pass a $\forall$/l step by a $\otimes$/l step, because $\forall$s will sometimes be subformulae of $\otimes$ formulae; thus the two types of rule applications must be intermixed.

So every derivation has a normal form. To prove that every derivation of a parsing problem sequent has a corresponding parse tree, we will need to put the derivation into normal form; we will then need to "index" the derivation in order to get indices for the parse tree.

Definition 4.8 Let $\mathcal{D}$ be a derivation in normal form which uses j applications of the Cont/l rule, and which has k bunches in the antecedent of its conclusion. Then the *indexing* of the derivation is arrived at by assigning indices to each bunch in each sequent of $\mathcal{D}$ in the following manner.

- Assign the bunches in the conclusion the indices $j+1, j+2, \ldots, j+k$, in that order.
- Assign the bunches introduced by each Cont/l step the indices $1, \ldots, j$, in the order in which they appear at the *top* of the sequence of Cont/l steps, propagating other indices from conclusion to premiss through those steps. (The bunches will be assigned indices so that the indices in the premiss of the topmost Cont/l step will be $1, \ldots, j$ in that order, with $j+1, \ldots, j+k$ possibly interspersed with them in that order.)
- Propagate indices from conclusion to premisses through the rest of the proof as follows:
  - Assign the bunches in the premisses of $\forall$/l, $\otimes$/l, and $\otimes$/r steps the same sequence of indices as in the conclusions.

– Assign the bunches in the premisses of $\leftarrow$/l and $\langle\rangle$/l steps the sequence of indices in the conclusion, excluding the index of the bunch being deleted.

Then the *analogue sequence of indexed parse trees* to any sequent S in an indexed derivation is as follows.

- If S is the axiom "$!\Gamma \vdash [\,]$", then its analogue sequence is the single indexed tree consisting of the node $[\,]$ with index m, where m is the highest index appearing in the entire derivation.
- If S is the axiom "$!\Gamma, \langle \mathbf{t} \rangle^i \vdash [\mathbf{t}]$", then its analogue sequence is the single indexed tree consisting of the node $[\mathbf{t}]$ with index i.
- If S is the conclusion of a $\otimes$/r application, the analogue sequence of its left-hand premiss is $T_1, \ldots, T_j$, and the analogue sequence of its right-hand premiss is $T_{j+1}, \ldots, T_k$, then the analogue sequence of S is $T_1, \ldots, T_k$.
- If S is the conclusion of a $\leftarrow$/l application, and is of the form $!\Gamma, \Delta, \langle \mathbf{t} \leftarrow \mathbf{B} \rangle^i, \Delta' \vdash \mathbf{t}$, and the analogue sequence of its premiss is $T_1, \ldots, T_k$, then the analogue sequence of S is the single tree consisting of the node $\mathbf{t}$ with index i and immediate subtrees $T_1, \ldots, T_k$.
- If S is the conclusion of a Cont/l, $\forall$/l, $\otimes$/l, or $\langle\rangle$/l application, then its analogue sequence is the analogue sequence of its premiss.

□

We are now in a position to prove the soundness theorem.

Theorem 4.9 Given a grammar G, if there is a derivation in Rsa (resp. Rsr) for the sequent

$$\mathcal{F}(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}$$

then the sequence of terminals $[\mathbf{t}_1] \cdots [\mathbf{t}_k]$ can be parsed as the non-terminal $\mathbf{s}$ under the tree admissibility (resp. rewriting) interpretation.

Proof. Consider the indexing of the normal form of the derivation; call this indexed derivation $\mathcal{D}$. Let the analogue parse tree of the conclusion of $\mathcal{D}$ be T (there will be only one tree in the sequence, because the consequent of the conclusion is the non-terminal $\mathbf{s}$). We can now prove several useful properties of T.

(1.) For every sequent in $\mathcal{D}$, every index appearing in its analogue sequence of trees appears as the index of a bunch. This is proven easily by induction.

(2.) More generally, for every sequent in $\mathcal{D}$, if the sequence of interior nodes with index i in the analogue sequence of trees is $\mathbf{t}'_1, \ldots, \mathbf{t}'_n$, and the sequence of children of each $\mathbf{t}'_j$ is $\mathbf{A}_{j1}, \ldots, \mathbf{A}_{jm_j}$, then there is a bunch in that sequent of the form $\langle (\mathbf{t}'_1 \leftarrow \mathbf{A}_{11} \otimes \cdots \otimes \mathbf{A}_{1m_1}), \ldots, (\mathbf{t}'_n \leftarrow \mathbf{A}_{n1} \otimes \cdots \otimes \mathbf{A}_{nm_n}) \rangle$. This is proven by induction as well; the only difficult case is the one in which a new clause is introduced.

(3.) For every sequent in $\mathcal{D}$, the sequence Λ of terminal leaf nodes appearing in the analogue sequence of parse trees (if Λ is non-empty) is exactly the sequence of elements of the antecedent bunch with the maximum index. This is easily proven by induction.

We can now prove the tree admissibility conditions on T. An interior node with index i corresponds to a $\leftarrow/\mathrm{l}$ step introducing a bunch with index i. By the method used to index bunches, no bunch with index i appears in the premiss of that step. By property (1), no node with index i therefore appears in the premiss's analogue sequence of trees. Therefore, the first condition (no node dominates any other node with the same index) is met.

The second condition (the local subtrees with index i correspond to a rule) follows from the facts that by property (2), the local subtrees of T with index i correspond to the bunch with index i in the conclusion; and that each bunch with index i corresponds to a rule in the grammar.

T is therefore an admissible parse tree.

In the special case that the derivation is in Rsr, we can also prove that T is a rewriting-consistent parse tree. The $\leftarrow/\mathrm{l}$ rule in Rsr introduces a bunch on the extreme left of the bunches in the antecedent; so by our indexing scheme, the index of the bunch introduced is lower than that of any of the other bunches in the sequent[4]. By property (1) above, therefore, *no* other node in the subtree corresponding to the conclusion of a $\leftarrow/\mathrm{l}$ step will have an index less than or equal to the root. T therefore satisfies the first condition on rewriting-consistent parse trees; since the other condition is identical to that of admissible parse trees, and T is admissible, T is therefore consistent with rewriting.

Finally, it is only necessary to note that since we can give an admissible (resp. rewriting-consistent) parse tree which, by (3) above, has a fringe with the sequence of terminals $\mathbf{t}_1, \ldots, \mathbf{t}_k$, that sequence can be parsed as $\mathbf{s}$ under the tree admissibility (resp. rewriting) interpretation.

□

To summarize the soundness and completeness results, a sentence is accepted by a particular SDG under the tree admissibility interpretation if and only if the corresponding sequent is derivable in Rsa, and under the rewriting interpretation if and only if the corresponding sequent is derivable in Rsr.

Since Rsa and Rsr are fairly general logical proof systems giving rules for connectives without special reference to the problem domain of SDGs (or computational linguistics in general), we can say that we have given a logical characterization of SDGs.

[4]There are pathological cases in which the bunch of terminals can appear to the left of a rule bunch, but these will not arise from a $\leftarrow/l$ rule. An example is $\langle s \leftarrow [a] \oslash b \rangle^1, \langle [a] \rangle^3, \langle b \leftarrow [\,] \rangle^2 \vdash s$.

4.5 Relationship to Other Relevance Logics

We should say a few words about how Rsa and Rsr compare to other relevance logics, particularly Intuitionistic Linear Logic, or ILL [GL87].

There is a simple translation of Rsa and Rsr sequents into ILL sequents: simply remove the "bunching" brackets $\langle\rangle$, and translate $\mathbf{A} \leftarrow \mathbf{B}$ into $\mathbf{B} \multimap \mathbf{A}$ (the linear logic implication) and $[\,]$ into I (the linear "true"). Given this translation, the rules of Rsa (and thus Rsr) are derivable rules of the sequent calculus of ILL; therefore, all theorems of Rsa and Rsr are theorems of ILL.

The commutativity of assumptions in ILL has a lot to do with this. The main way in which Rsa and Rsr differ from ILL is that the order of assumptions within bunches is important (although in Rsa the order of bunches is unimportant). However, Rsa and Rsr are both more powerful than "non-commutative ILL", as it is usually formulated. That is because in non-commutative ILL one is not allowed to permute assumptions, *and* the $\otimes$/r rule must split the *entire* list of assumptions down the middle:

$$\otimes/\mathrm{r}: \qquad \frac{!\Gamma \vdash \mathbf{B} \quad !\Gamma' \vdash \mathbf{B}'}{!\Gamma, \Gamma' \vdash \mathbf{B} \otimes \mathbf{B}'}$$

The Rsa/Rsr $\otimes\!\!\!/$r rule, which allows some formulae from *each* bunch to go to each premiss, is therefore not allowed.

The concept of bunching has not to our knowledge been discussed in the context of linear logic; that concept comes rather from more mainstream relevance logic [Rea88]. There, there are the concepts of intensional bunching (in which members of bunches cannot be permuted) and extensional bunching (in which they can). However, the $\otimes$/r rule still splits the antecedent in the middle, so the Rsa/Rsr rule is more powerful here as well.

The Rsa/Rsr bunches, then, though they have some similarity to relevance-logic bunches, would seem to be of a novel kind. They are naturally explained as capturing information to do with sequencing of events, and do not take us out of the realm of the accepted, completely commutative systems. However, their novelty does lead us to hope that some better-established way of encoding sequencing could be used in the proof systems, or that other applications could be found for this style of bunch.

5 Conclusions

We have given a relevance logic characterization of SDG parsing which elegantly captures linguistic features such as the significance of temporal word sequencing and the undesirability of redundant elements.

It is interesting to note that this characterization turns out to be in agreement with the main principles of relevance theory in cognitive sciences [SW86]. In particular, this theory postulates that an individual's cognitive goal at a given moment is always an instance of the more general goal of maximizing the relevance of the

information processed, and that an assumption which has no contextual effect in a given context is irrelevant in that context. In our parsing as deduction context, it is clear that duplicate assumptions, as well as added, unnecessary assumptions, should be disallowed as irrelevant, and that the information conveyed through word ordering should be appropriately accounted for as pertinent.

Thus, in a modest scale, our work uncovers and partly formalizes a natural link between inferential theories of communication, relevance logic and logic programming, which was already informally advocated in 1987 by R. Kowalski[5]. His contention that logic programming as a field should be looking more closely at linguistic and cognitive science paradigms has led, more recently, to a certain identification between (appropriate subsets of) natural language and logic programming itself [Kow90].

Further areas of research include the uses of relevance linguistics in categorial grammars, and in formalizing SDG accounts of Immediate Dominance/Linear Precedence (ID/LP), where order is also an important concept. It may be possible to translate a categorial grammar straightforwardly into an extension of the SDG formalism; interestingly, the extensions that are needed seem to involve iterated implication and other concepts from Miller et al.'s work [MN86, HM91]. With respect to ID/LP parsing, it may be possible to extend the SDG framework to allow for the more fine-grained conditions on word ordering needed in ID/LP; this would presumably be reflected in the way in which permutation was handled in the logical characterization.

6 Acknowledgements

Thanks to Dale Miller and Valeria de Paiva for their papers and subsequent correspondence, and to Greg Restall for discussions on relevance logic. The relevance logic sequent calculus is due to Andrews; the presentation is due to all three authors.

This research is supported by Veronica Dahl's operating grant (OGP0002136) from the Natural Sciences and Engineering Research Council of Canada (NSERC), and by Fred Popowich's research fellowship from the BC Advanced Systems Institute, and his NSERC operating grant (OGP0041910). We are grateful to the Logic and Functional Programming Group, in whose lab this work was developed, and to the Centre for Systems Science, Laboratory for Computer and Communications Research and School of Computing Science at Simon Fraser University for the use of their facilities.

References

[AB75] Alan R. Anderson and Nuel D. Belnap. *Entailment: The Logic of Relevance and Necessity.* Princeton University Press, Princeton, N.J., 1975.

[5]Banquet Speech, Fourth International Logic Programming Conference, Melbourne, Australia

[Bol91] A. W. Bollen. Relevant logic programming. *Journal of Automated Reasoning*, 7:563–585, 1991.

[Dah89] Veronica Dahl. Discontinuous grammars. *Computational Intelligence*, 5:161–179, 1989.

[DM90] Veronica Dahl and Pierre Massicotte. Interpreting static discontinuity grammars in Prolog. *Applied Mathematics Letters*, 3, 1990.

[DP90] Veronica Dahl and Fred Popowich. Parsing and generation with static discontinuity grammars. *New Generation Computing*, 8:245–274, 1990.

[DP91] Valeria De Paiva. A Dialectica model of the Lambek calculus. In *8th Amsterdam Logic Colloquium*, Amsterdam, December 1991.

[Gen69] Gerhardt Gentzen. *The Collected Papers of Gerhard Gentzen*. North-Holland, Amsterdam, 1969. Ed. M. E. Szabo.

[GH69] S. Greibach and J. Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences*, 3:233–247, 1969.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GKPS85] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Blackwell, Oxford, 1985.

[GL87] Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In *Theory and Practice of Software Engineering*, 1987.

[HM91] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. In *Logic in Computer Science*, Amsterdam, July 1991.

[HS84] Masami Hagiya and Takafumi Sakurai. Foundation of logic programming based on inductive definition. *New Generation Computing*, 2:59–77, 1984.

[HSH90] Lars Hallnäs and Peter Schroeder-Heister. A proof-theoretic approach to logic programming I: Clauses as rules. *Journal of Logic and Computation*, 1(2), 1990.

[Kow90] Robert Kowalski. English as a logic programming language. In *UK Association for Logic Programming Conference*, Bristol, March 1990.

[Lam88] J. Lambek. Categorial and categorical grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, volume 32 of *Studies in Linguistics and Philosophy*, Dordrecht, 1988. D. Reidel.

[Llo84] John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1984.

[MN86] Dale A. Miller and Gopalan Nadathur. Higher-order logic programming. In *Proceedings of the Third International Logic Programming Conference*, Imperial College, London, July 1986.

[Per81] Fernando Pereira. Extraposition grammars. *American Journal for Computational Linguistics*, 7:243–256, 1981.

[PW80] Fernando Pereira and David H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[Rea88] Stephen Read. *Relevant Logic.* Blackwell, Oxford, 1988.

[SW86] Dan Sperber and Deirdre Wilson. *Relevance: Communication and Cognition.* Harvard University Press, Cambridge, Massachussetts, 1986.

[Yet90] D. Yetter. Quantales and (non-commutative) linear logic. *Journal of Symbolic Logic*, 55, March 1990.

Centre for Systems Science

• Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.

•CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.

• CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.